A Generic Architecture For Local Computation

Master Thesis of Marc Pouly

Department of Informatics
University of Fribourg

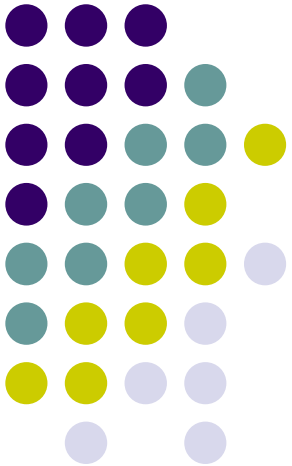
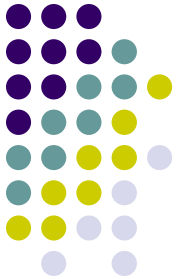
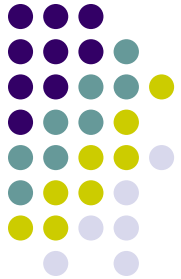


Table of Contents



- Goals & Requirements
- Realizing a Valuation Algebra
- Realizing Local Computation & Join Trees
- Join Tree Construction
- Demonstration
- Conclusions
- Future Work

Goals & Requirements I



A generic architecture for local computation unifies two basic concepts:

1. Different instances of a valuation algebra
2. Different architectures of local computation

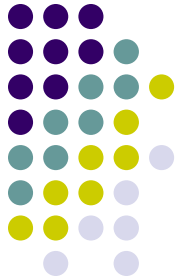
Both of them are somehow generic concepts:

- A valuation algebra due to it‘s mathematical definition (3 operations & 6 axioms).
- Different architecture types differ in their message passing algorithm and in the appropriate join tree type.

An important difference between those two concepts is:

- Instances of a valuation algebra are very large in number.
- There exists 4 relevant architectures of local computation.

Goals & Requirements II



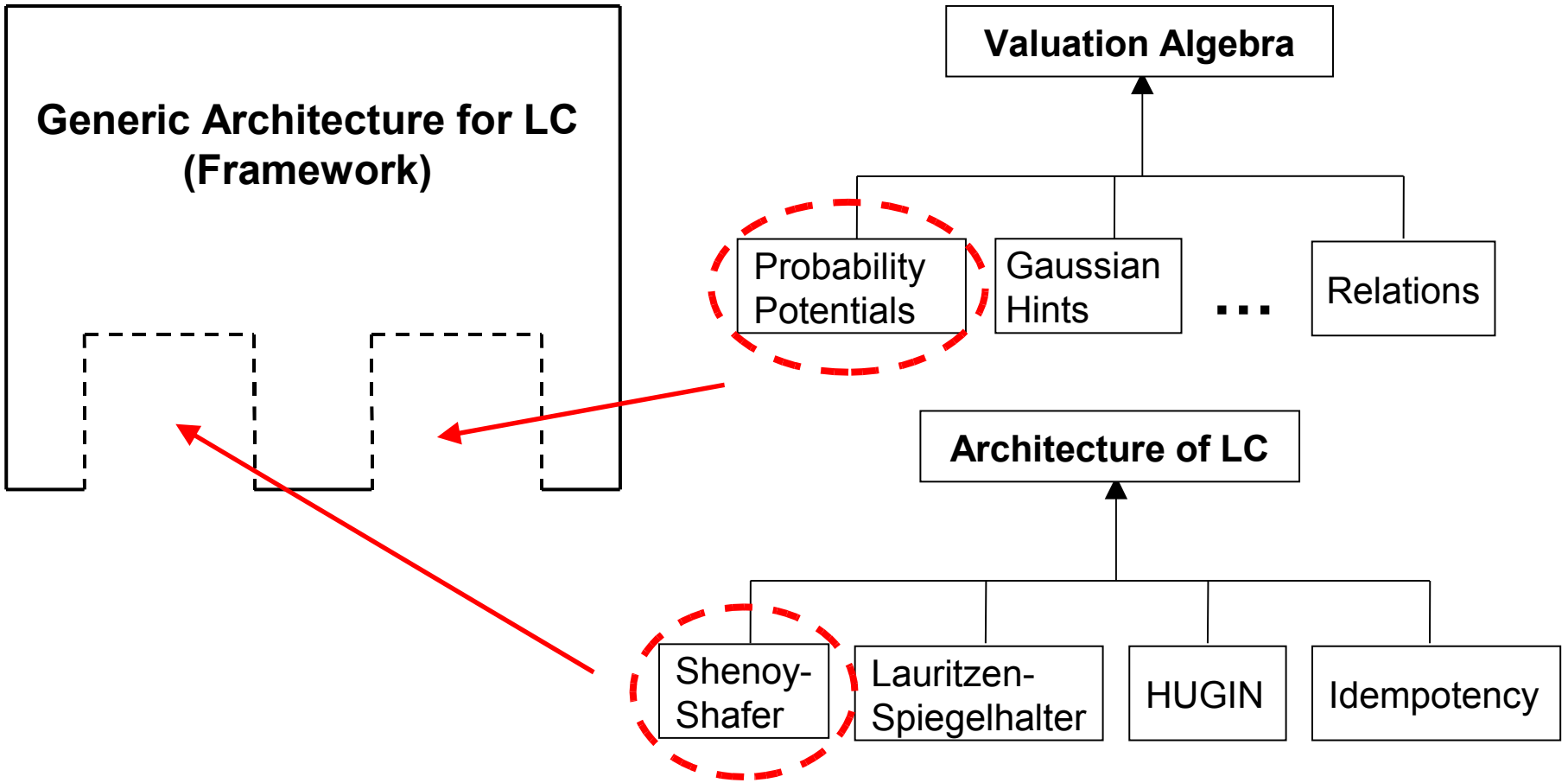
A generic architecture for local computation (LC) must provide solutions for the following requirements:

- The user can himself implement new instances and integrate them in the whole architecture.
- Different instances can be combined with different architectures of LC (satisfying the math. restrictions).
- Architectures for LC are interchangeable (existing code stays untouched).
- System for educational motivations.

Two catchwords:

- Unit construction system
- Framework technology

Goals & Requirements III



Valuation Algebra I

A Valuation Algebra is defined by 3 operations (labeling, combination & marginalization) and 6 axioms.

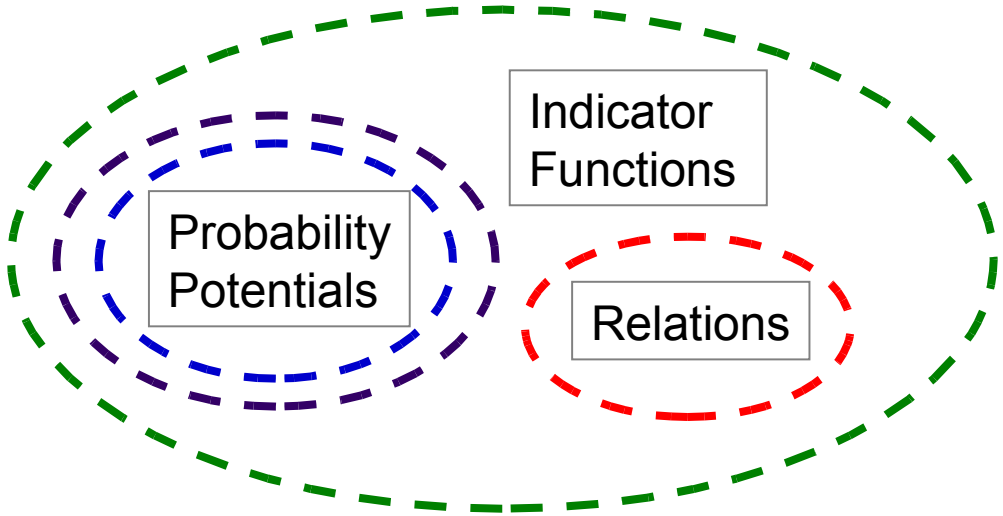
But they may have further properties being important in the context of LC.

Mathematical properties:

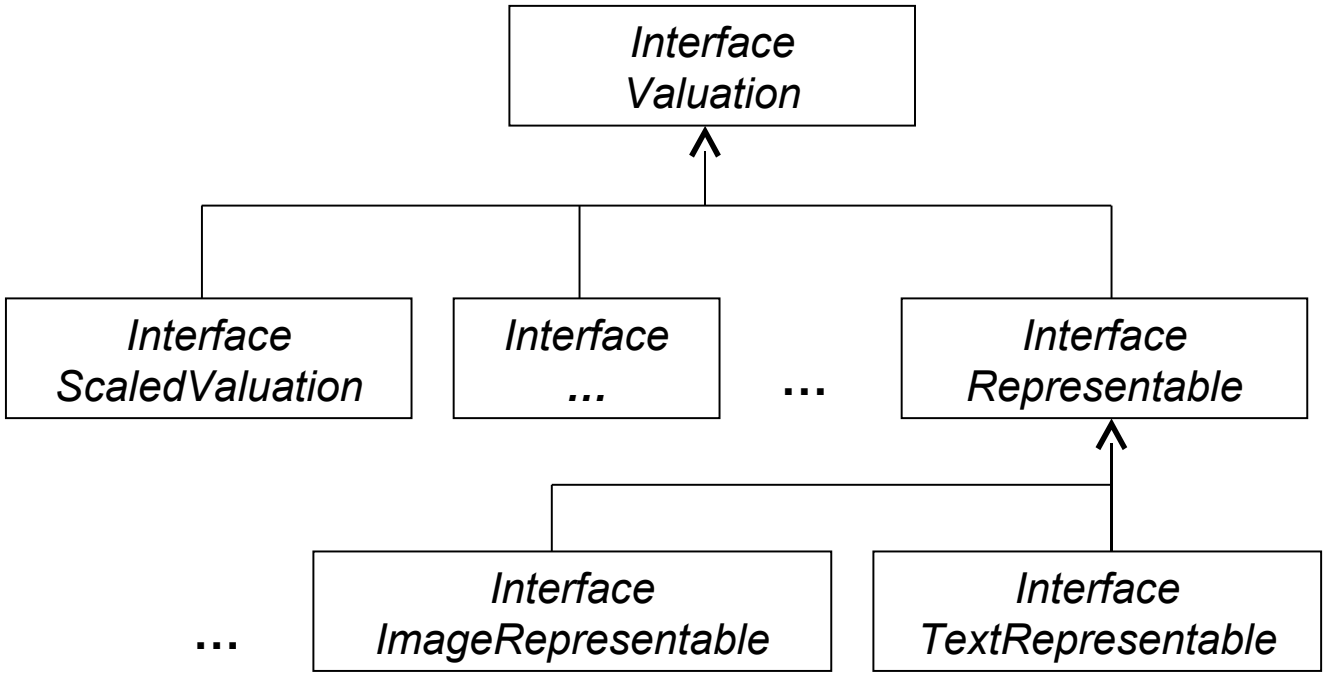
- Idempotency
- Scaling
- ...

Non-mathematical properties:

- Text Representation
- Graphical Representation
- ...



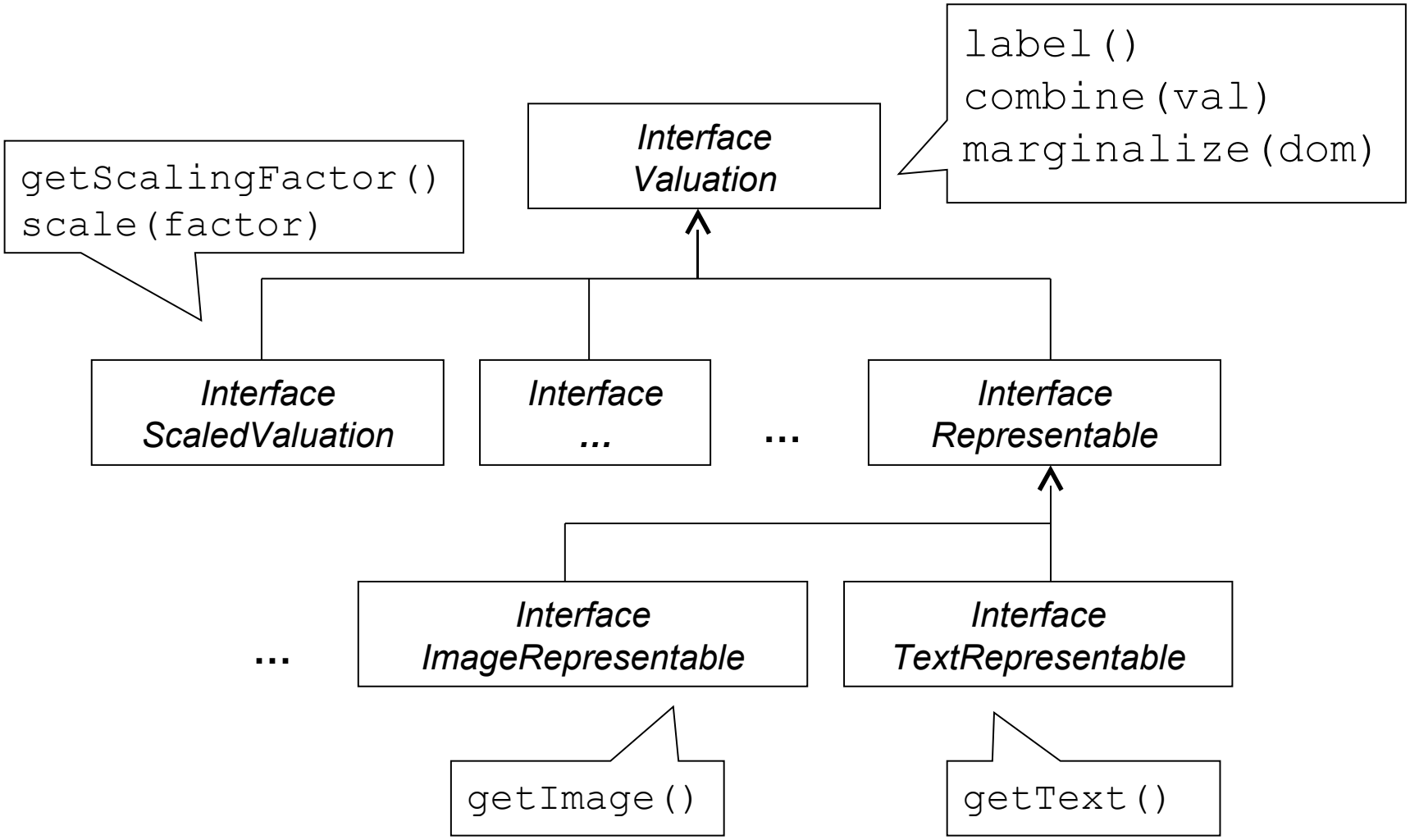
Valuation Algebra II



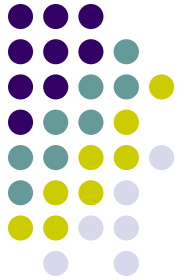
Advantages of this system of interfaces:

- allows any combination of properties
- allows several representations of a single instance
- is extendable (other implementations stay untouched)

Valuation Algebra III



Valuation Algebra IV



Beside valuations, the other components of a valuation algebra are:

- Variables → Interface (Identity check)
- Domains (set of Variables) → pre-implemented set type

To sum it up, a valuation algebra is fully implemented by:

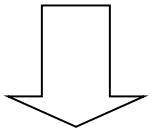
- A class implementing the Variable interface.
- A class implementing the Valuation interface.

Local Computation Overview

Rem: Different architecture of LC differ in their message passing algorithm and in the appropriate join tree type.

Join trees are basically linked sets of nodes (standard tree implementation).

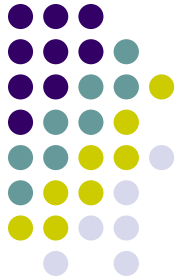
Once a join tree is defined, one can realize the appropriate message passing algorithm as a recursive method onto its‘ nodes.



The most fundamental concept of LC are join trees. To garantuee that architectures of LC are interchangeable, we need an abstract join tree type.

The concept of LC demands doubly-linked join tree nodes (inward & outward propagation).

Join Tree I



An abstract join tree type contains the following informations:

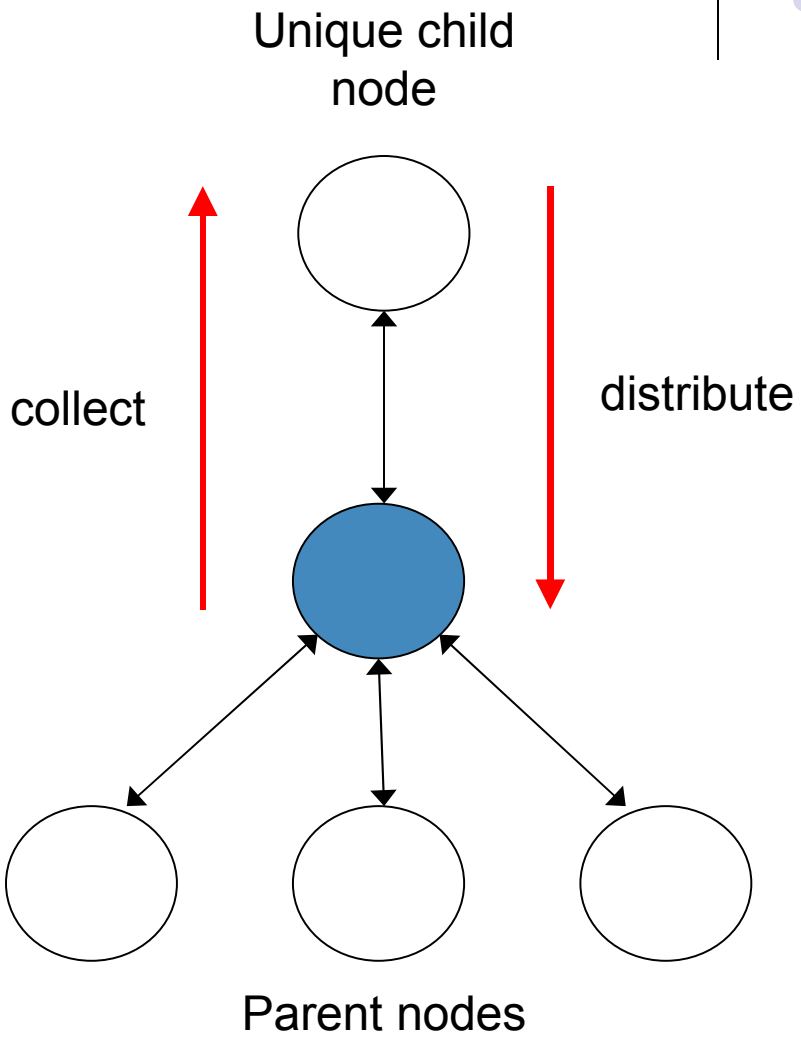
- A pointer onto its root node (tree implementation).
- An array of pointers onto the leaves (to start collect phase).
- Infos for educational purposes (ex: elimination sequence, …).

This requires an abstract (doubly-linked) node type for a full realization of a tree data-structure. These nodes are the skeleton of a join tree and are basically built up from the following components:

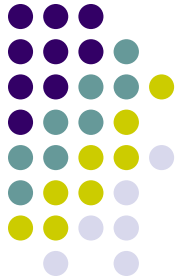
- A pointer onto its unique child node.
- An array of pointers onto its parent nodes.
- A domain (Markov property).

Join Tree II

This abstract node type is represented by an interface. But the architecture offers also a pre-implemented node type for n-ary nodes.



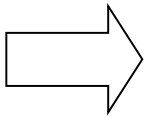
Join Tree Construction I



Thoughts about join tree construction:

- Join trees are constructed by a slightly modified fusion algorithm.

- The construction of join trees is widely independent from the join tree architecture. Whenever a new node is introduced, we choose the node type according to the architecture.

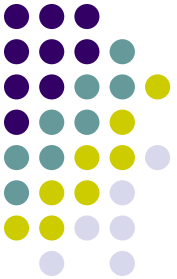


The join tree construction is pre-implemented and hidden from the user.

Problem: How does a join tree know the node type corresponding to its architecture?

Another point is that no additional overhead should arise, when the user wants to construct multiple join trees of the same architecture type but with different input valuation sets.

Join Tree Construction II



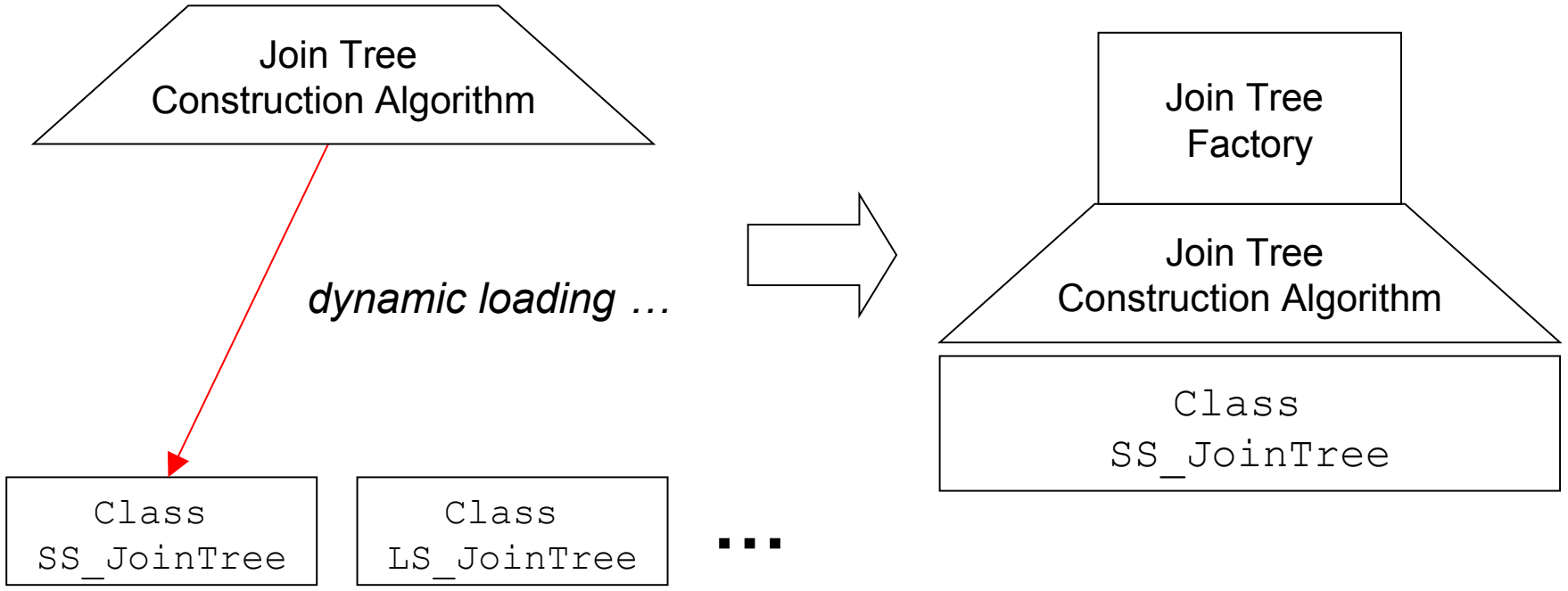
A solution is presented by a so-called join tree factory.

- The user creates a new join tree factory. The constructor demands the class name of the appropriate join tree class.
- This factory will load the given join tree class. The class itself contains a static method getNode() which creates a new node whose type corresponds to the join tree‘s architecture.
- To build a join tree, the user calls the factory‘s `create(…)` method with the initial set of valuations as argument.
- The factory delivers a new join tree object.
- The user starts the inward & outward propagation onto this join tree object.

Join Tree Construction III

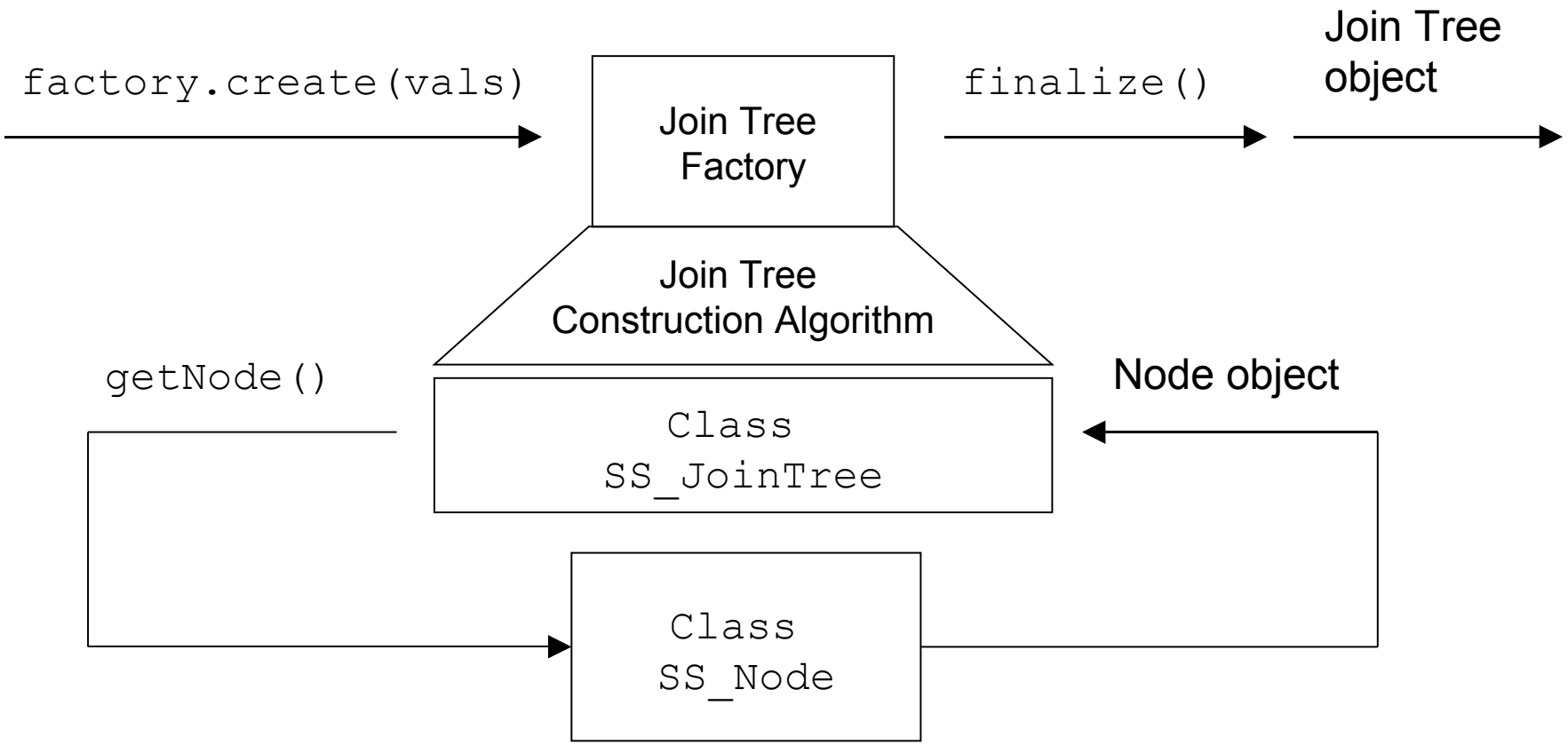
Example: Creating a Shenoy-Shafer join tree factory:

```
JoinTreeFactory f = new JoinTreeFactory("SS_JoinTree");
```



Join Tree Construction IV

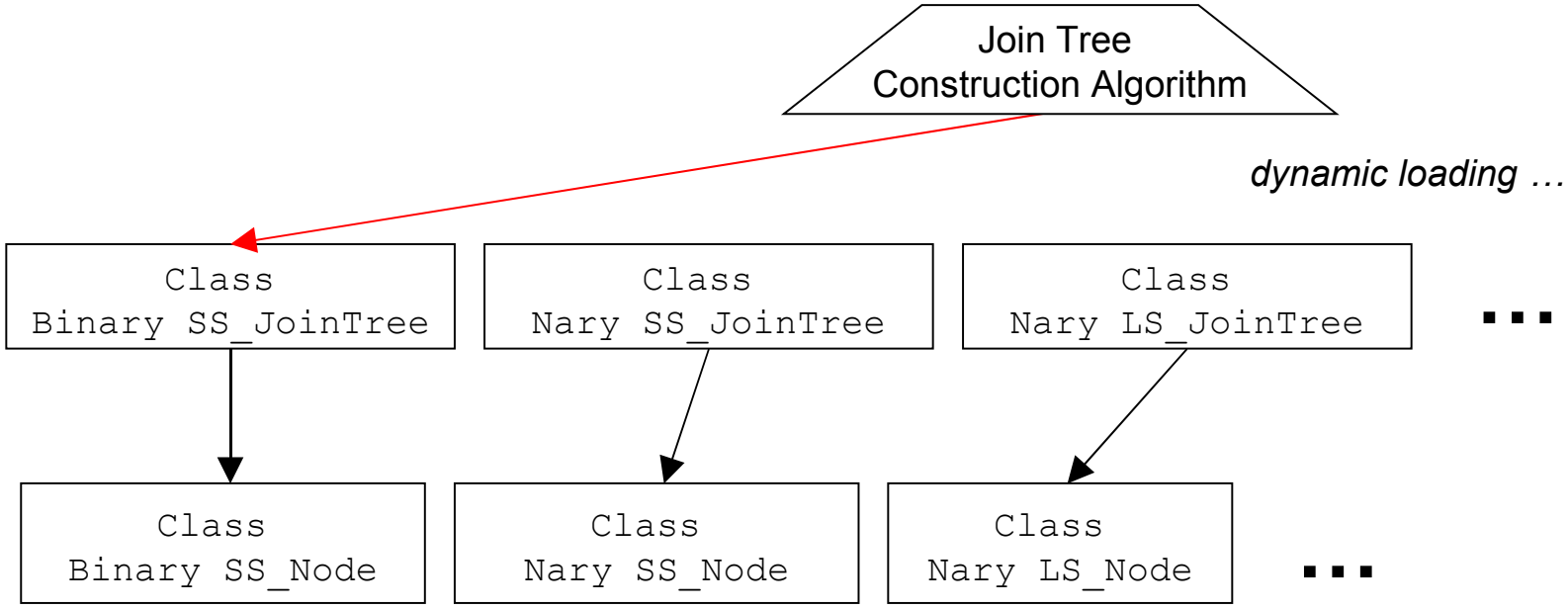
Example: Creating a new Join Tree:



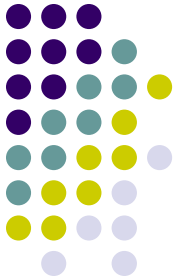
Join Tree Construction V

Why dynamic loading:

- Construction algorithm is independent from the architecture.
- Possible to have multiple construction algorithms.
- Changing architecture type by modifying one single path (string).
- Although there are only 4 architectures of LC, we can make further differentiations on node level.



Local Computation I



All architectures of LC can be separated in an inward & outward propagation phase. The implementation of those two algorithms is made on the node level.

Inward propagation (bottom-up):

- The leaves send their messages directly to their unique child nodes.
- As soon as the current node got a message from each of its parents, it computes the new message and sends it to the unique child node. The algorithm is applied resursively onto the child node.
- Stop when the root node has received all incoming messages.

Local Computation II

Outward propagation (top-down):

- The root node sends it‘s messages directly to all parents.
- As soon as the current node got a message from its child, it computes the new messages and sends them to all parent nodes. The algorithm is applied resursively onto all parents.
- Stop when the leaves have received the incoming message.

Query processing (top-down):

- If the current node‘s domain satisfies the query, return its valuation.
- Else, apply the algorithm recursively on all parent nodes.

Demo

1. Probability potentials are implemented as valuation algebra.

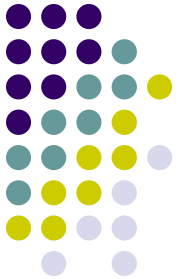
3. Variables are simple strings and each variable knows its frame.

5. Constructor of a potential demands an array of variables and an array of probabilities. The potential (combining configurations with probabilities) is generated automatically.

7. A binary version of the Shenoy-Shafer architecture is used.

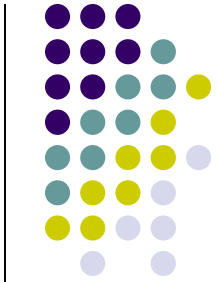
9. The input values are taken from the ASIA example. (Lauritzen-Spiegelhalter 88).

Conclusions

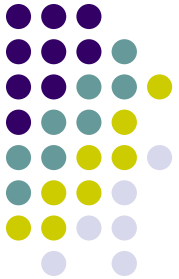


- The abstract implementation of a valuation algebra is an exact realization of the underlying mathematical concept.
- The generic implementation of LC architectures has to prove its worth.
- Meeting all requirements of the different architecture types was the most challenging part.
- Framework for educational purposes → no optimisation.
- JoinTree Viewer only for small applications.

Future Work



- Implementing all other architectures of LC.
- Implementing new valuation algebra instances.
- Swapping join tree construction algorithm.
- Optimization and profiling
- Parallel / Distributed computing
- Extending the JoinTree Viewer to a GUI



Questions ?